Cloud Computing, REST and Mashups to Simplify RFID Application Development and Deployment

Conference Paper · January 2011

DOI: 10.1145/1993966.1993979 · Source: DBLP

3 authors, including:

Sanjay E. Sarma
Massachusetts Institute of Technology
272 PUBLICATIONS **10,840** CITATIONS

SEE PROFILE

Cloud Computing, REST and Mashups to Simplify RFID Application Development and Deployment

Dominique Guinard
Inst. for Pervasive Computing
ETH Zurich
and MIT Auto-ID Labs
Massachusetts Inst. of Technology
dguinard@mit.edu

Christian Floerkemeier
Inst. for Pervasive Computing
ETH Zurich
and MIT Auto-ID Labs
Massachusetts Inst. of Technology
floerkem@mit.edu

Sanjay Sarma
MIT Auto-ID Labs
Massachusetts Inst. of Technology
sesarma@mit.edu

ABSTRACT

While of increasing importance for the real-time enterprise, deployments of Internet of Things infrastructures such as RFID remain complex and expensive. In this paper, we illustrate these challenges by studying the applications of the EPC Network which is an RFID standards framework that aims to facilitate interoperability and application development. We show how the use of blueprints that were successful on the Web can help to make the adoption of these standards less complex. We discuss in particular how Cloud Computing, RESTful interfaces, Real-time Web (Websockets and Comet) and Web 2.0 Mashups can simplify application development, deployments and maintenance in a common RFID application. Our analysis also illustrates that RFID/EPC Network applications are an excellent playground for Web of Things technologies and that further research in this field can significantly contribute to making real-world applications less complex and cost-intensive.

1. INTRODUCTION

The RFID (Radio Frequency IDentification) standards community has developed a number of air interfaces and software standards to provide interoperability across RFID deployments. This extensive standards framework, known as the EPC (Electronic Product Code) Network, covers aspects such as reader-to-tag communication, reader configuration and monitoring, tag identifier translation, filtering and aggregation of RFID data, and persistent storage of application events. While there are in total fifteen standards that make up the EPC Network framework, the air interface protocol known as EPCglobal UHF Gen2 has seen the most adoption – both in large scale supply chain applications as well as niche RFID deployments.

The adoption of the software standards within the EPC Network has been significantly slower. The deployment of RFID applications that implement the EPC Network standards often remains complex and cost-intensive mostly because they involve the deployment of often rather large and heterogeneous distributed systems. As a consequence, these systems are often only suitable for big corporations and large implementations and do not fit the limited resources of small to mid-size businesses and small scale applications both in terms of required skill-set and costs.

While there is most likely no universally available solution to these problems, the success of the Web in bringing complex, distributed and heterogeneous systems together through the use of simple design patterns appears as a viable approach to address these challenges. The contribution of this paper is a discussion of pain points in RFID applications that have made deployments challenging and a proposal how they can be addressed using solutions directly inspired from the architecture of the Web and its services. Our work also illustrates the applicability of Web of Things [6] concepts to the world of RFID applications.

This paper is structured as follows. We begin by describing a typical use-case of the EPC Network. We then look at three important pain points. For each pain point we propose and implement a Web blueprint (i.e., architectural pattern) and discuss the respective related work. Finally, we illustrate the benefits by means of two concrete prototypes and discuss the challenges we encountered.

1.1 Case Study: Electronic Article Surveillance with RFID

In this section, we describe a common RFID application, RFID as an electronic article surveillance technology, and illustrate how the EPC Network framework can be used to realize this application. This example will help us illustrate the challenges in RFID application development and deployments in later sections of this paper.

In many clothing stores, RFID technology is set to replace existing Electronic Article Surveillance (EAS) technology because of its many advantages. The two most important issues include knowledge about the product being stolen and the reduction in the number of false alarms. Today, retail stores have little information about which particular product is actually being stolen. As a consequence, the stores cannot replenish shelves appropriately resulting in a possible lost sale to a consumer who is willing to pay for the item. There is also no way to prevent frequent false alarms where products with active EAS tags from another retailer trigger the alarm.

WoT 2011, June 2011; San Francisco, CA, USA

The use of RFID technology as an EAS system relies on the presence of RFID tags on the individual clothing items as well as RFID readers at the backroom door to the store, at the checkout and at the exit. When products are placed on the shop floor, the RFID tags are read as they pass the reader at the backroom store entry. The applications registers the tags and marks the IDs as 'on sale'. If a consumer decides to purchase an item, the RFID tag is read again at the checkout and flagged as 'sold'. If the user leaves the store with the product, the RFID readers at the exit report the RFID tag to the application, but no alarm is triggered because the product is marked as sold. If the consumer decides to leave the store without paying, the RFID tag is identified by the readers at the exit and an alarm is triggered because the product is still not paid for.

To realize the above example, the RFID readers need to be mounted in the store and connected to a local area (wireless) network. After discovery of the readers on the network, each RFID reader needs to be configured to read RFID tags and report RFID tag read reports via the binary EPCglobal LLRP reader protocol. To prevent RFID readers running continuously, the backroom reader is often triggered via a motion sensor. There is also an alarm connected to the network that can be triggered by the RFID readers at the exit. Following the configuration of the readers, an application server needs to be set up on a server in the clothing store that runs the RFID middleware. In the case of the EPC Network, such an application server would run an instance of an Application-Level-Events (ALE) compliant middleware that filters and aggregates the RFID data. Using the ALE WS-* API, the developer would need to group the RFID readers at the various locations (entry, exit, and checkout) and also define time filters and aggregators that eliminate redundant RFID reads. In a typical RFID deployment, the appearance of an RFID tag in the read range of an RFID reader can result in numerous tag reads of the same tag. To process the filtered and aggregated RFID data, custom business logic needs to be implemented. The business logic of the EAS application needs to deserialize the incoming ALE SOAP messages containing the tag reads, to send off web service EPCIS (EPC Information Services) query interface to check the state of the particular tag ('on sale', 'sold'), and to create a new EPCIS event that triggers a state change and possibly sound an alarm. To store and access these EPCIS events, the developer needs to set up a database on the application server and deploy an EPCIS repository that supports the EPCIS capture and query protocols. The developer might also decide to develop a custom application that queries the EPCIS repositories across multiple stores to provide analytics capabilities. The retailer might for example want to identify the products most stolen and locations of stores with the most stores.

2. PAIN POINTS AND REMEDIES

A direct consequence of the complexity of installing and implementing the use-case we described before is that many smaller businesses decide to adopt very basic solutions where non-standard tags simply trigger an alarm everytime they pass the gate. In this section, we analyze some of the pain points of RFID application development, deployment and maintenance that are illustrated by the above scenario.

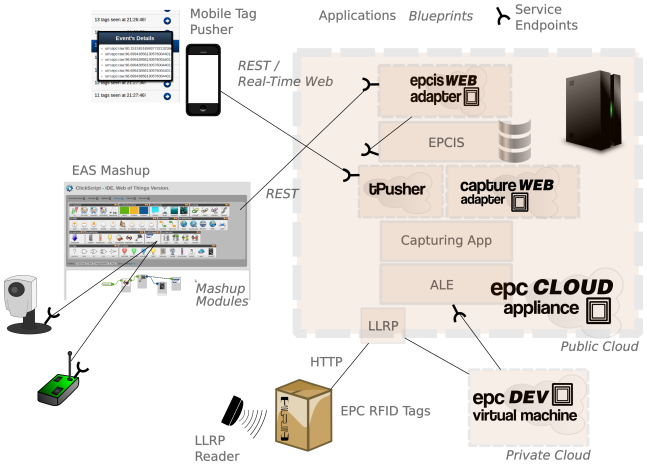

Figure 1: Overview of the EPC Cloud component architecture

2.1 Pain Point: Complex and Expensive Backend Deployment and Maintenance

Real-world, industrial IoT systems often encompass several relatively complex standards and involves several different software components [9]. The EPC Network is no exception; Vendors of EPC products offer several different software components often sold separately to form an EPC software stack. As an example, the Fosstrak open-source project[1] implements most of the EPC standards and requires the installation of 9 different software components in order to be able to run a standard use-case, such as the EAS one described before. Additionally, a full Fosstrak installation also requires a compatible Java SDK, Apache Maven, a full MySQL database and an Apache Tomcat server, summing up the number of required software components to 13. As a consequence, a full EPC software stack is rather complex to install and deploy and *often requires software experts*, especially when considering businesses for which IT is not a core concern (e.g., operators in the supply chain) or smaller businesses. The complexity is further increased by the maintenance work required by a number of different components and their respective updates and patches cycles. Hence, deploying and *maintaining IoT systems is time consuming* and accounts for more than 70% of the system's overall software costs [1].

Furthermore, the software components often need to be deployed on application servers running on dedicated hardware. For the Fosstrak stack, a Java Application Server (or a least a servlet container such as Tomcat) is required which needs to be configured on a hardware server to handle the appropriate load and accesses. Similarly, the IoT embedded devices (e.g., RFID readers, sensor nodes, etc.) need to be deployed, maintained and configured. This induces *significant hardware costs and the need for hardware experts.*

2.1.1 Remedy: A Cloud-Based Virtual Infrastructure

Virtualization Blueprint. Reducing complex software installation is one value proposition of virtualization platforms [7]

[1] `http://www.fosstrak.org`

such as VMWare[2] or the open-source Virtual Box[3]. With these platforms, software stack can be installed once in a virtualized OS (operating system) called virtual machine or guest OS, and then shared to be deployed within minutes on any supported host machine running the virtualization platform. This significantly reduces the installation costs and required skills.

In the IoT space, this benefit has been identified and is increasingly used by several platforms such as the Instant Contiki virtual machine[4] which offers a complete development environment for WSNs (Wireless Sensor and Actuator Networks) ready to use within minutes [10]. However, the EPC Network still lacks such solutions. Hence, we virtualized an EPC software stack. The EPC Dev Virtual machine combines a Linux Ubuntu Operating System, with an Eclipse IDE (Integrated Development Environment), a source repository (Maven), as well as an Apache Tomcat container in which we deployed and configured the 9 Fosstrak software components. This means that the virtual machine can be used as a development environment or as a test server instance of the EPC software stack if installed on an appropriate server machine.

This cuts down the installation time of a full EPC software stack from several hours or days to a few minutes. It further fosters quick evaluation of a complete EPC software stack which can be of great help when assessing different implementations or developing proof of concept prototypes or enhancements of the EPC software stack.

Cloud Computing: Utility Computing Blueprint. While Virtualization significantly reduces installation time, it does not solve two other pain points of IoT deployments: software and hardware maintenance costs. However, recent developments in the Web 2.0 and especially the trend towards providing services on the Web rather than simply Web-pages, have led to a convergence of virtualization technologies and the distributed Web, leading to Cloud Computing.

Cloud Computing can take several forms under the umbrella of two big groups. "Private Clouds" are basically virtualized environments running locally as described in the previous section. "Public Clouds" are, on the other hand virtualized environments running on remote machines. A Public Cloud can take many forms [12]. In its "Utility Computing" form, it basically proposes to further push the notion of virtualization by making the hardware on which virtual machines run available as a virtual resource pool fully accessible, on-demand, on the Web. Amazon Web Services (AWS)[5] pioneered the space of Utility Computing followed by many others such as IBM, Microsoft, Rackspace and VMWare. Recently, Cloud Computing has been increasingly used in conjunction with WSNs [2] as a way to reduce complexity.

We experimentally applied the Utility Computing blueprint to the EPC software stack using the AWS platform and in particular the EC2 service. Amazon EC2 allows the creation and management of virtual machines (Amazon Machine Images, or AMIs) that can then be deployed on demand onto a pool of machines hosted, managed and configured by Amazon. We created a server-side AMI, called EPC Cloud Appliance based on Linux Ubuntu Public Cloud edition[6] and containing the 13 software components required by a full installation of Fosstrak.

This concretely means that any company or research institution willing to deploy an EPC software stack can simply log onto AWS, look for the EPC Cloud AMI, select the type and number of remote servers it should be deployed on. Once the virtual servers are running (which typically takes less than 5 minutes), an RFID reader can be connected. If the reader does not offer a Web-management interface or a default configuration, the Fosstrak LLRP Commander and its Eclipse-based UI is available in the EPC Dev Virtual machine and can be used for configuring it. Then, the readers are described by accessing the configuration offered in Web UI of the EPC Cloud Appliance. Once this is done, the cloud instance will contact the reader and start recording the tag reads.

A direct benefit of the approach is that the server-side hardware maintenance is delegated to the cloud provider which is often more cost-efficient for smaller businesses [12]. Furthermore it also offers better scaling capabilities as the company using the EPC Cloud AMI, can deploy to additional and more powerful instances within a few clicks from the Web front-end (or Web API) of AWS and will be charged only for the resources they actually use.

2.2 Pain Point: Complicated Applications Developments

The idea behind most commercial IoT deployments is the integration of real-world data to business systems or end-consumer applications. This requires to interface existing or new applications with the IoT infrastructure. Thanks to the recent advent of smart phones, companies are also increasingly willing to create mobile applications using IoT deployments.

In the case of the EPC network, the application integration point is the EPCIS standard. While the EPCIS provides a simple and lightweight HTTP interface for recording EPC events, its query interface is a standardized WS-* (i.e., SOAP, WSDL, etc.) interface. WS-* applications are complex systems with high entry barriers and require developer expertise in the domain which is often an issue when considering small to mid-size businesses. Moreover, WS-* are often not well adapted to more light-weight and ad-hoc application scenarios [8] such as mobile or Web applications.

2.2.1 Remedy: RESTful Architectures and Real-Time Web

RESTful Web services are based on a Representational State Transfer (REST) [3] architecture. REST uses the Web as an application platform and fully leverages all the features inherent to HTTP such as browser access, scalability and caching, authentication and encryption. In projects often unified under the umbrella of "Web of Things", REST is used and adapted for real world devices (e.g., WSNs, appliances or tagged objects, etc.) in order to create a "universal API for things" [6].

When compared to WS-*, RESTful Web services are rather lightweight [8, 11] and directly usable from a browser and with well-known Web languages (e.g., JavaScript, HTML, PHP, Python, etc.) which lowers the entry barrier for de-

[2] http://www.vmware.com
[3] http://www.virtualbox.org
[4] http://www.sics.se/contiki/instant-contiki.html
[5] http://aws.amazon.com
[6] http://www.ubuntu.com/cloud/public

velopers. Furthermore, the Web currently accounts for one of the most active pools of developers and as a consequence finding Web developers is easier for companies (and especially small to mid-size companies) than finding highly skilled embedded systems experts. Thus, REST results in a general simplification of the development process [8] for ubiquitous use-cases such as mobile or Web applications.

Hence, we propose to create a Resource Oriented Architecture for the EPC Network and offer two RESTful APIs for building applications consuming RFID data.

RESTful Business EPC Events: EPCIS Webadapter. The first RESTful API meets the needs of mobile, Web, or WSN clients wanting to get business-level RFID events, thus, it is deployed on top of the EPCIS. As a client, the EPCIS offers an interface to query for RFID events. This interface is accessible through a WS-* Web Service. While this enables users to create clients using several languages supporting Web services, it makes it impossible to directly query for RFID events using Web languages such as JavaScript or HTML. More importantly it does not allow for exploring the EPCIS using a Web browser, searching for tagged objects or exchanging links pointing to traces of tagged objects. Thus, we implemented a RESTful pluggable module for any standard compliant EPCIS called the EPCIS Webadapter[7] and further described in [5].

Clients of the Open Source EPCIS Webadapter such as browsers or Web applications can query for tagged objects directly using the uniform HTTP interface. Requests are then translated into WS-* calls on the standard EPCIS interface. The direct benefit of the EPCIS Webadapter is that every RFID event, reader, tagged-object or location is turned into a Web resource and gets a globally resolvable URI which uniquely identifies it and can be used to retrieve various representations. Thus EPCIS queries are transformed into compositions of these identifiers and can be directly executed in the browser, sent by email or bookmarked. As an example, a factory manager who wants to know what tagged objects enter his factory can bookmark a URI like: `http://.../epcis/rest/location/urn:company:factory1/reader/urn:company:entrance:1`

Real-Time Web: Pushing from Readers to Web Clients. The second RESTful API meets the need for mobile or Web clients to access the raw data directly pushed by RFID readers through the LLRP and ALE protocols. The challenge here is that the Web was designed mainly as a client-pull architecture, where clients can explicitly request (pull) data and receive it as a response. This makes uses-cases where near real-time communication is required rather challenging. As an example, a typical use-case is one in which we would like to push events that are being recorded by an RFID reader directly to a mobile browser application for monitoring purpose (see Section 3.1 for an implementation of this use-case).

Here, the "Real-time Web", one of the most recent blueprints of the Web, can be leveraged. The Real Time Web encompasses several new techniques that can be used to push events directly to browsers. We focus on two of these here. The first one, called Comet (also called HTTP streaming or long-polling) is based on the concept of long-lasting HTTP connections and keep-alive messages. While this is supported by most browsers and HTTP libraries, it works by using an existing loop-hole. More recently, Web-sockets (part of the HTML5 drafts) were proposed. Websockets propose duplex communication with a single TCP/IP connection directly accessible from any compliant browser through a simple Javascript API. The increasing support for HTML5 in Web and Mobile Web browsers makes it a very good candidate for pushing data on the Web.

For the EPC Network, we created two components as shown on Figure 1. The Capture App Webadapter is a modular Web application which gets events from ALE and redirects them to a number of RESTful Services (e.g., to the EPCIS Webadapter) for further processing. The services the application sends the events to can be configured through a RESTful interface on the Web as well, which allows to flexibly decide where RFID events should be routed to.

The second component is called tPusher and combines a RESTful API with a Web-socket and Comet server. Using a RESTful API, clients can subscribe to RFID event notifications for a particular reader by using a URL such as:
`http://.../t-pusher/reader/READER_ID`
This initiates a Web-socket connection with the server on which RFID events recorded by `READER_ID` will be pushed.

Our implementation is based on Atmosphere[8], a Java abstraction framework for enabling push support on most Java Web servers. One of the advantages of this approach is to be able to deploy tPusher on recent Web Servers such as Grizzly[9], which are highly optimized to push events on the Web because of their usage of non-blocking threads for each client. In order to support browsers or other clients that do not support HTML5 Websockets yet, we use a client-side abstraction Javascript library called Atmosphere JQuery Plugin which falls back to a Comet connection in case Websockets are not supported by the client.

2.3 Pain Point: Tedious Business Case Modeling and Cross-IoT Systems Integration

RFID use-cases generally do not involve RFID readers and tags only – they are usually combined with sensors and actuators. In the EAS system we described before, several sensors (e.g., RFID readers, motion sensors) and actuators (e.g., alarm) are combined together to form the basic use-case. These combinations of RFID, sensors and actuators often occur at a low level, sometimes even at the wiring level. This mainly has two drawbacks. First it requires to combine the complicated and often not homogeneous low-level APIs of (expensive) devices which requires expert knowledge. Then, once installed, these compositions of devices are static and cannot be flexibly reconfigured to integrate new sensors or actuators.

2.3.1 Physical Mashup Platforms

Web 2.0 Mashups are a recent form of Web applications that compose several Web APIs using scripting languages to create new composite applications. Since in the Web of Things, every device is accessible through a Web (RESTful) API, physical mashups have been proposed [6]. These applications combine WoT devices with each other and with virtual service on the Web. Core to physical mashup is the

[7] `http://www.webofthings.com/rfid`

[8] `http://atmosphere.java.net`

[9] `http://grizzly.java.net`

notion of a mashup module. A module is a software abstraction that accepts a few inputs and delivers an output. Modules can be composed or "piped" together to form a use-case. Typically this occurs in a Mashup Editor. Figure 3 shows a visual representation of modules in the Clickscript Mashup Editor[10]. Once a mashup has been successfully created and tested locally using a Mashup Editor it should be deployed to a Mashup Engine in the cloud such as the Physical Mashup Framework [4] where is it going to be executed remotely.

Thanks to the blueprints we applied before, we can take a similar approach to create composite applications in the EPC Network. We create several simple modules that users of the EPC Cloud can then compose to create their use-cases. Our current implementation supports the following EPC-related modules:

- RFID Reader: Inputs: business location (e.g., exit-gate), URL. Output: matching EPCs.
- EPCIS: Inputs: EPC, business step (e.g., checkout), URL. Output: true or false.
- CaptureApp and tPusher: Inputs: String to push, URL of the push endpoint. Output: true or false.

It is worth noting that thanks to the RESTful APIs each module is a simple Javascript snippet that fits within 20 lines of code.

3. PROOF OF CONCEPT EVALUATION BY PROTOYPING

While a quantitative evaluation is an important part of our future work, we tested the EPC Cloud by building two proof of concept prototypes, focusing on how the pain points can be relaxed by using the proposed Web blueprints.

3.1 Mobile TagPusher

When setting up RFID readers or maintaining existing deployments it is valuable to have a direct feedback of the tags observed by a particular reader in order to monitor the manufacturing process or to debug the readers. In the current implementations of the EPC software stack this would require to use and configure a monitoring tool such as the Fosstrak LLRP Commander on a desktop computer. Thanks to the *RESTful interface* of the Capture App Webadapter as well as the Real-Time Web capability of tPusher, the tags observed by any reader can now be directly pushed to any browser or HTTP library.

Because these events are of interest in-situe, we developed as Mobile Web page that can display them in a user-friendly manner. The page uses HTML5 and Javascript with the Atmosphere JQuery Plugin we described before. All code required for such a page to subscribe to events pushed by readers through the Capture App Webadapter and display them fits within 5 lines of Javascript. The code is shown below:

```
1 //called whenever an event is pushed:
2 function callback(response) {alert(
      response.responseBody + response.
      transport);}
3 //subs. to the events of reader "exit1"
4 $.atmosphere.subscribe(
5   "http://EPC_CLOUD_APPLIANCE/capture-
        webadapter/reader/exit1",
6   callback, $.atmosphere.request = {
        transport: 'websocket' });
```

[10] http://clickscript.ch

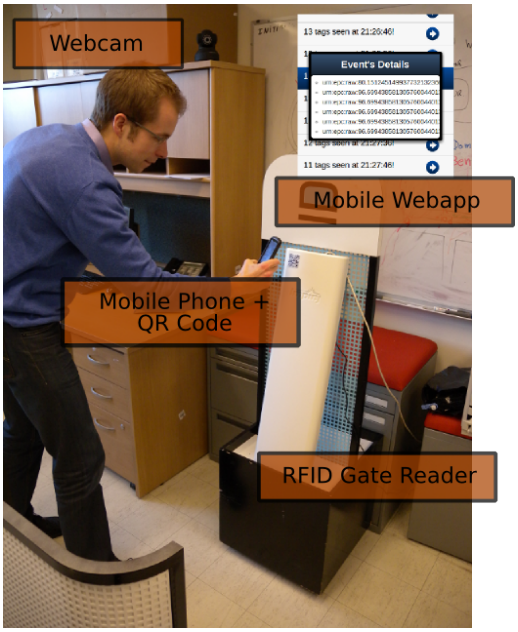

Figure 2: Real-time Web push from an RFID reader to a mobile browser

As shown on Figure 2, we deployed this prototype in a lab environment. Each reader features a QR-Code containing its unique URL in the EPC Cloud. When scanning this tag with a mobile phone it redirects the user to the HTML5 Web page shown on the top right corner of Figure 2. As tags are read by the readers, the Web page automatically receives and displays new events. We successfully tested this prototype on Android and iOS devices.

3.2 EAS Mashup

With this prototype we illustrate how the concept of *physical mashups* can be applied to the EPC Network towards more flexible and cheaper EAS systems. As shown in Figure 2, our lab setting reflects the exit gate of a store. We place an RFID gate at the exit, monitored by a cheap off-the-shelf IP-enabled Webcam[11].

We then compose these elements into a use-case within a few clicks using a version of the Clickscript mashup editor accomodated to support WoT type of devices [4]. For this, we combine three main modules with some basic language constructs as shown on Figure 3. From the RFID Reader module we get events pushed whenever a tag is read by the "exit-gate" reader. The EPCIS module is then used to check whether the tag just read went through the business-step "checkout". If it is not the case, we trigger the Webcam module which takes a picture. Finally, the tPusher module pushes the URL of the picture to any listening client application (i.e., a mobile Web application in our case).

[11] http://www.foscam.com

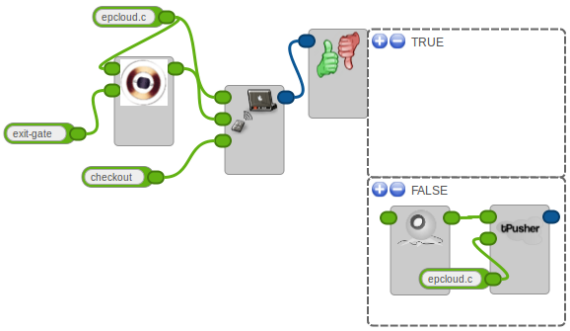

Figure 3: The EAS Mashup modeled with the Clickscript Mashup Editor

4. DISCUSSION AND FUTURE WORK

While working on the proposed approach and its prototype we identified a number of challenges and discuss three of them here. First, while some standard LLRP readers offer a reader-initiated scheme, most operate on a server-initiated scheme. This means that the EPC Cloud server has to contact the RFID readers in order to start the reading process. While this works fine in places where a direct access to the Internet is available, this is problematic in industrial environments where RFID readers sit behind firewalls and do not feature public IP addresses. To solve this problem, LLRP standard readers should also offer a reader-initiated scheme.

Second, to optimize data access, most cloud infrastructures offer highly optimized storage services (e.g., NoSQL databases) that can be easily distributed and load balanced. In the Java world, the implementations of these services is compliant with the Java Data Object (JDO) which abstracts from the actual storage service being used and also allows to easily switch the service. Unfortunately, the current Fosstrak EPCIS is not JDO compliant but uses JDBC and is rather tightly coupled with a MySQL database. Porting the EPCIS to JDA would enable to better leverage the scalability that cloud solutions have to offer.

Finally, for real-world applications, network delays might be serious drawback as events and actions are sent and triggered in the cloud. While it is unlikely that an EPC Cloud solution will support true real-time use-cases in the near future, our early measurements have shown typical delays of less than a second on average for the described EAS Mashup, from the reader, to the cloud and then to the mobile phone. While this is acceptable for most envisioned applications we still need to extensively evaluate the proposed blueprints and their optimized implementations in real-world deployments.

5. CONCLUSION

In this paper we have shown how Web blueprints and patterns can be beneficial to IoT infrastructures. In particular we looked at the EPC Network and explained how virtualization, cloud computing, REST and the real-time Web as well as the concept of physical mashups could contribute to a wider adoption of the EPC Network standards and tools. Virtualization allows us to package all the development tools into a single virtual machine that can be run virtually anywhere. Cloud computing simplifies deployment and maintenance of the EPC software stack. Thus, we push the standardized EPC Network closer to small and mid-size businesses that could benefit from it. By taking a resource oriented approach and using the real-time Web, we can offer more lightweight interfaces and allow innovative mobile, Web and WSN applications to directly use the EPC Network. Finally, by offering a mashup editor and engine we allow more flexible simple use-cases, where existing sensors and actuators can be directly integrated from the Web with RFID hardware.

6. REFERENCES

[1] R. D. Banker, S. M. Datar, C. F. Kemerer, and D. Zweig. Software complexity and maintenance costs. *Communications of the ACM*, 36:81–94, Nov. 1993. ACM ID: 163375.

[2] J. Bungo. Embedded systems programming in the cloud: A novel approach for academia. *Potentials, IEEE*, 30(1):17–23, 2011.

[3] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Techn.*, 2(2):115–150, 2002.

[4] D. Guinard. Mashing up your web-enabled home. In *Adjunct Proc. of ICWE 2010 (International Conference on Web Engineering)*, Vienna, July 2010.

[5] D. Guinard, M. Mueller, and J. Pasquier. Giving RFID a REST: building a Web-Enabled EPCIS. In *Proc. of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, Nov. 2010.

[6] D. Guinard, V. Trifa, and E. Wilde. A resource oriented architecture for the web of things. In *Proc. of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, Nov. 2010.

[7] K. L. Kroeker. The evolution of virtualization. *Communications of the ACM*, 52:18–20, Mar. 2009. ACM ID: 1467253.

[8] C. Pautasso and E. Wilde. Why is the web loosely coupled? a Multi-Faceted metric for service design. In *Proc. of the 18th International World Wide Web Conference (WWW'09)*, Madrid, Spain, Apr. 2009.

[9] T. Riedel, N. Fantana, A. Genaid, D. Yordanov, H. Schmidtke, and M. Beigl. Using web service gateways and code generation for sustainable IoT system development. In *Proc. of Internet of Things 2010 International Conference (IoT 2010)*, Tokyo, Japan, 2010.

[10] J. J. P. C. Rodrigues and P. A. C. S. Neves. A survey on IP-based wireless sensor network solutions. *International Journal of Communication Systems*, 2010.

[11] D. Yazar and A. Dunkels. Efficient application integration in IP-based sensor networks. In *Proc. of the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings (BuildSys)*, Berkeley, CA, USA, Nov. 2009.

[12] S. Zhang, S. Zhang, X. Chen, and X. Huo. Cloud computing research and development trend. In *Proc. of the International Conference on Future Networks, 2010. ICFN '10*, pages 93–97, 2010.